*TMS320 DSP*
*DESIGNER'S NOTEBOOK*

# *Interfacing a TMS320C3x to the TLC320AD58C 18-Bit Stereo A/D Converter*

*APPLICATION BRIEF: SPRA271*

*Martin Staebler*
*Digital Signal Processing Products*
*Semiconductor Group*

*Texas Instruments*
*July 1996*

TEXAS INSTRUMENTS

## IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

**CONTACT INFORMATION**

| | |
|---|---|
| US TMS320 HOTLINE | (281) 274-2320 |
| US TMS320 FAX | (281) 274-2324 |
| US TMS320 BBS | (281) 274-2323 |
| US TMS320 email | dsph@ti.com |

# Contents

**Abstract** ..... 7
**Design Problem** ..... 8
**Solution** ..... 8

# Figures

**Figure 1. TLC320AD58C Serial Interface 18-bit Master Mode '100' Timing Diagram** ..... 8
**Figure 2. TLC320AD58C-to-TMS320C3x interface** ..... 10

# Tables

**Table 1. Master-clock-to-sample-rate Conversion** ..... 9

# Examples

**Example 1. TMS320C3x 'C' Program Listing** ..... 11
**Example 2. C3x.h, header file listing** ..... 13
**Example 3. Vectors.h, TMS320C3x Interrupt Vector Table Listing** ..... 14

# Interfacing a TMS320C3x to the TLC320AD58C 18-Bit Stereo A/D Converter


## Abstract

This document discusses how to initialize the TMS320C3x to interface with the TLC320AD58C 18-bit stereo analog-to-digital converter (ADC) without glue logic.

The TLC320AD58C serial interface provides several master and slave modes for 16-bit or 18-bit data output, to be compatible with a wide range of DSPs. To interface with the TMS320C3x 32-bit floating-point DSP, the 18-bit master mode ‘100’ was chosen to get an 18-bit resolution result and meet the TMS320C3x serial port requirements. This document gives timing diagrams, a schematic and several code listings to allow a designer to readily implement this solution.

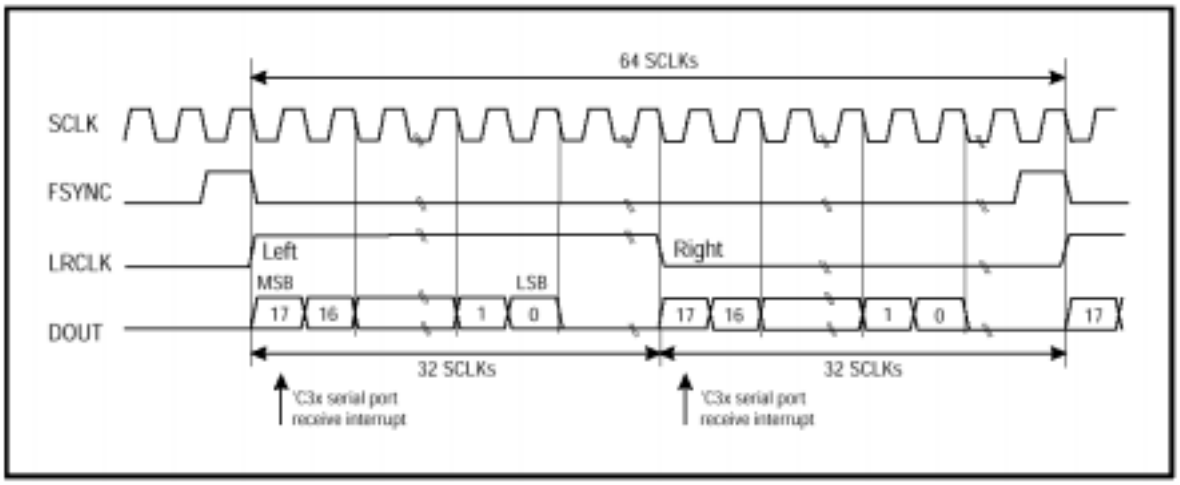
## Design Problem

How do I initialize the TMS320C3x to interface with the TLC320AD58C 18-bit stereo analog-to-digital converter (ADC) without glue logic?

## Solution

The TLC320AD58C serial interface provides several master and slave modes for 16-bit or 18-bit data output to be compatible with a wide range of DSPs. To interface with the TMS320C3x 32-bit floating-point DSP, the 18-bit master mode ‘100’ was chosen to get an 18-bit resolution result and meet the TMS320C3x serial port requirements. The timing diagram is shown in Figure 1.

*Figure 1. TLC320AD58C Serial Interface 18-bit Master Mode ‘100’ Timing Diagram*




The frame sync signal (FSYNC) is then used to designate valid data from the ADC, and is active for one shift clock period. After the falling edge of FSYNC, the left channel data is shifted out on the falling edge of SCLK with the most significant bit (D17) first. When the last data bit is shifted out, the output remains low for another 14 SCLKs to get a total of 32 SCLK periods each channel. After 32 SCLKs, LRCLK goes low and the right channel data is then shifted out, respectively. FSYNC and LRCLK frequency are fixed to the sampling frequency ($F_S$ = MCLK/256 or MCLK/384 depending on the status of the CMODE input pin). The conversion cycle is synchronized to the rising edge of LRCLK, thus to the falling edge of FSYNC. Although data is shifted out in two separate time packets representing the left and right channel digital output, the analog inputs are sampled and converted simultaneously. In the master mode, SCLK, FSYNC, and LRCLK are generated internally from

MCLK depended on the status of the CMODE input pin, as shown in Table 1.

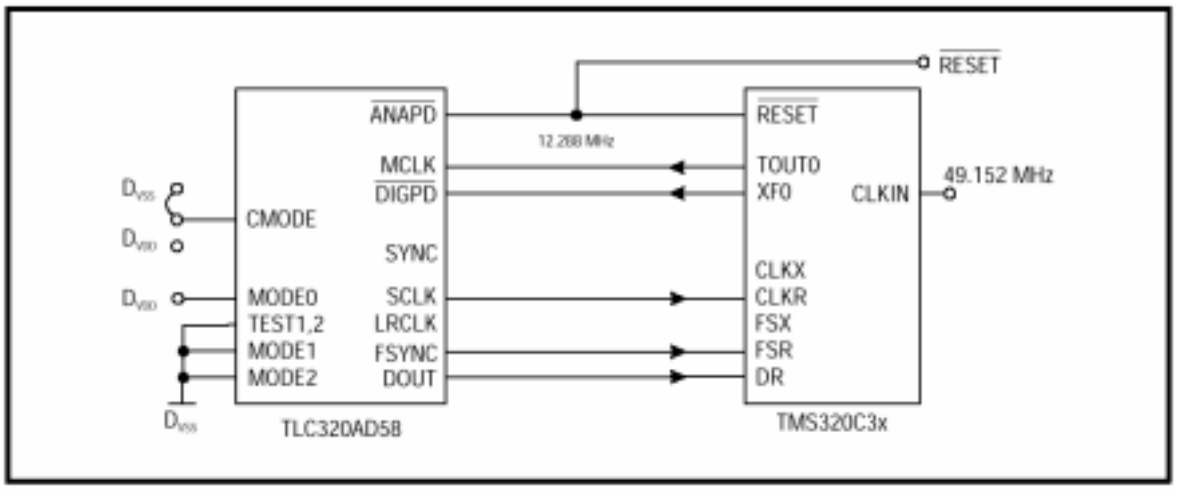
*Table 1. Master-clock-to-sample-rate Conversion*

| MCLK (MHz) | CMODE | SCLK (MHz) | Sample Rate (kHz) |
|---|---|---|---|
| 12.288<br>18.432 | Low<br>High | 3.072 | 48 |
| 11.2896<br>16.9344 | Low<br>High | 2.822 | 44.1 |
| 8.129<br>12.288 | Low<br>High | 2.048 | 32 |
| 0.256<br>0.384 | Low<br>High | 0.064 | 1 |

The TMS320C30 employs two bidirectional serial ports while the TMS320C31 and TMS320C32 each have one. Each serial port controls six port pins: FSR/FSX, CLKR/CLKX, and DR/DX for receiving/transmitting data, respectively. Figure 2 shows the glueless interface to the TLC320AD58C utilizing the SCLK, FSYNC, and DOUT signals. Mode ‘100’ is set by pulling MODE1 and MODE2 pins low and MODE0 pin high. The master clock is derived from the TMS320C3x to make sure all clock signals are synchronized. The TMS320C3x is running at 49.152 MHz and provides the required MCLK frequency of 12.288 MHz at the timer 0 output pin in order to get a 48-kHz sample rate. CMODE has to be pulled low. If other sample rates are required, refer to Table 1.

The TLC320AD58C analog function blocks are initialized together with the DSP by a system reset, after all supply voltages are stable. The digital function blocks are initialized by pulling down /DIGPD for some μs. After the rising edge of /DIGPD, the device resumes normal operation. When /DIGPD is low, the TLC320AD58C digital function blocks are shut down and power consumption is reduced. However, if power down mode is not required, this signal can be tied to /ANAPD. In both cases, refer to the TI Data Acquisition Circuits Data Book (SLAD001) for setup timing requirements. All digital inputs and outputs of the TMS320C3x and the TLC320AD58C are 5-V TTL compatible. To reduce ringing and overshot, a serial damping resistor (50 Ω) is recommended for the master clock signal.

*Figure 2. TLC320AD58C-to-TMS320C3x interface*




The TMS320C3x can be configured to receive a maximum of 32 bits of data per word. The TLC320AD58C transmits a total of 64 bits after the FSYNC pulse appears. To force the DSP to read the left and right channel, the TMS320C3x serial port configuration is toggled between continuous mode and burst mode. In burst mode, FSYNC indicates the start of a new data transfer. In continuous mode, the new data transfer starts immediately after the last bit of the previous transfer has been shifted out. Both the serial port and the timer registers are memory mapped. Eight memory-mapped registers are provided for each serial port:

- one global control register - defines the serial port configuration
- two control registers - set the function of the CLKX/R, FSX/R, and CLKX/R pins
- three receive/transmit timer registers
- one data receive register
- one data transmit register

If the serial port shift clock (CLKR/CLKX) is generated externally, the corresponding timer can be used as a general-purpose timer. Refer to the TMS320C3x User's Guide (SPRU031) and the TMS320C32 Addendum (SPRU132) for more information on the TMS320C3x serial port.

**TMS320C3x Software Example**

The following TMS320C' program initializes the TLC320AD58C and the TMS320C30 serial port 1 to meet the TLC320AD58C serial interface timing requirements, and sets up the timer 0 period register

to generate the required MCLK frequency. On a serial port 1 receive interrupt, which occurs after receiving 32 bits from either the left channel or right channel, the program reads from the serial port receive register and converts the input signal into a floating-point number within –1.0 and 1.0. It then changes the serial port configuration from burst to continuous mode when the right channel has been received, or from continuous to burst mode when the left channel has been received. The transmit port is configured as the receive port for connection to the 18-bit TMS57014A stereo DAC. Remember that the data has to be written to the data transmit register no later than three CLKX cycles before the FSYNC pulse occurs (burst mode) or the next transfers starts (continuous mode).

*Example 1. TMS320C3x 'C' Program Listing*

```
/*****************************************************************/
/* File: AD58.C */
/* Interfacing the 18-Bit TLC320AD58 to TMS320C3x */
/*****************************************************************/

/* include files */
/*---------------*/
#include   "vectors.h"
#include   "c3x.h"

/* global variables */
/*------------------*/
float      l_channel;
float      r_channel;
/*---------------------------------------------------------------*/
/*                main program                                    */
/*---------------------------------------------------------------*/
void main(void)
{
   asm("    ldi        1000h,ST");   /* clear and enable cache */
   asm("    ldi           0h,IE");   /* clear all interrupt masks */
   asm("    ldi           0h,IF");   /* clear all pending interrupt */
   init_t0();                       /* Generate AD58 MCLK, if required */
   init_s1();                       /* Initialize serial port 1 */
   init_ad58();
   asm("    ldi _ERINT1_CPU,IE");   /*enable serial port 1 receive int*/
   asm("    or       _GIEBIT,ST");  /* global enable interrupts */

   while(1);                        /* wait on interrupt */
}

/*---------------------------------------------------------------*/
/* Subroutine to initialize Serial Port 1 to                     */
/*  communicate with TLC320AD58                                  */
/*---------------------------------------------------------------*/
void init_s1(void)
{
   serial_port[1][X_PORT] = X1_MODE;
   serial_port[1][R_PORT] = R1_MODE;
```

```
   serial_port[1][GLOBAL] = S1_CONFIG;
}

/*-------------------------------------------------------------------*/
/* Subroutine to initialize Timer 0 to generate TLC320AD58 MCLK    */
/*-------------------------------------------------------------------*/
void init_t0(void)
{
   timer[0][GLOBAL] = T0_HOLD;
   timer[0][T_COUNTER] = 0x0;
   timer[0][T_PERIOD] = T0_PERIOD;
   timer[0][GLOBAL] = T0_HOLD;
}

/*-------------------------------------------------------------------*/
/* Serial Port Receive Interrupt Service Routine                     */
/*-------------------------------------------------------------------*/
void c_int08(void)
{
/* reconfigure serial port to receive both channels                  */
/*   within one frame sync                                          */
   if (serial_port[1][GLOBAL] & 0x0C00)
   {
       /* read LEFT channel and normalize within -1.0..1.0 */
       l_channel = ((float) (serial_port[1][R_DATA] >> 14))/(4.0*65536);

       /* switch to burst mode */
       serial_port[1][GLOBAL] = serial_port[1][GLOBAL] & 0xFFFFF3FF;

       /* if transmitting to DAC, make sure to write to the transmit
            register no later than 3 SCLK=CLKX cycles before the rising
            edge of FSYNC */
   }
   else
   {
       /* read RIGHT channel and normalize within -1.0..1.0 */
       r_channel = ((float) (serial_port[1][R_DATA] >> 14))/(4.0*65536);

       /* switch to continuous mode */
       serial_port[1][GLOBAL] = serial_port[1][GLOBAL] | 0x0C00;

       /* if transmitting to DAC, make sure to write to the transmit
             register no later than 3 SCLK=CLKX cycles before the next
             transfer */
   }
}
/*-------------------------------------------------------------------*/
/* Subroutine to initialize TLC320AD58 */
/*-------------------------------------------------------------------*/
void init_ad58(void)
{
   asm(" ldi 0010b,IOF"); /* reset XF0, power down AD58 */
   asm(" rpts 2500 "); /* wait for 100 usec before */
   asm(" nop "); /* asserting DigPwd */
   asm(" ldi 0110b,IOF"); /* AD58 normal operation */
}
```

*Example 2. C3x.h, header file listing*

```
/*-------------------------------------------------------------------------*/
/* FILE: C3X.H                                                             */
/* TMS320C3X CONTROL REGISTER SETTINGS TO SETUP INTERFACE WITH             */
/* TLC320AD58 - 18 BIT MASTER MODE                                         */
/*-------------------------------------------------------------------------*/

/*------------------------------*/
/* Serial Port 1 Initialization */
/*------------------------------*/
#define X1_MODE 0x000000111      /* FSX/DX/CLKX are serial port pins */
#define R1_MODE 0x000000111      /* FSX/DX/CLKX are serial port pins */
#define S1_CONFIG 0x00EBC3C00    /* Serial-Port Configuration */
                                 /* FSX/FSR input */
                                 /* FSX/FSR signals active high */
                                 /* external CLKX/R */
                                 /* CLKX/CLKR active low */
                                 /* fixed data rate mode */
                                 /* 32-bit data width */
                                 /* TX/RX interrupts are enabled */
                                 /* XRESET/RRESET set to 0 */
                                 /* (take out of reset) */

/*------------------------*/
/* Timer 0 Initialization */
/*------------------------*/
/* TOUT Frequency (clock mode) = 1/[8*CLKIN*T0_PERIOD], if           */
/*                                                  T0_PERIOD period>0 */
/*                              = 1/[4*CLKIN], if T0_PERIOD period = 0 */
#define T0_PERIOD 0        /* TOUT0 = 12,288 MHz for 49.152 MHz CLKIN */
#define T0_HOLD   0x0301   /* clock mode, 50% duty cycle */
#define T0_GO     0x03C1

/*----------------*/
/* Interrupt Mask */
/*----------------*/
asm("_ERINT1_CPU .set  80h");   /* enable serial port 1 receive int */
asm("_GIEBIT     .set  2000h"); /* global enable interrupts */

/*-------------------------------------------------------------------------*/
/* TMS320C3X CONTROL REGISTER LOCATIONS                                    */
/*-------------------------------------------------------------------------*/

/*--------------*/
/* Serial Ports */
/*--------------*/
/* SERIAL PORT BASE LOCATION */
volatile int (*serial_port)[16] = (volatile int (*)[16]) 0x808040;

/* SERIAL PORT CONTROL REGISTERS */
#define GLOBAL   0               /* GLOBAL CONTROL */
#define X_PORT   2               /* TRANSMIT CONTROL */
#define R_PORT   3               /* RECEIVE CONTROL */
#define X_DATA   8               /* TRANSMIT DATA */
#define R_DATA  12               /* RECEIVE DATA */
```

```
/*-------*/
/* Timer */
/*-------*/
/* TIMER BASE LOCATION */
volatile int (*timer)[16] = (volatile int (*)[16]) 0x808020;
#define T_COUNTER 4
#define T_PERIOD 8
```

*Example 3. Vectors.h, TMS320C3x Interrupt Vector Table Listing*

```
/*-------------------------------------------------------------------*/
/* Filename: vectors.h   Defines interrupt vectors and trap vectors */
/*                       for C programs                              */
/*                                                                   */
/* Usage:    #include vectors.h                                      */
/*                                                                   */
/* Modifications:  If you add interrupt service routines, modify   */
/*                 this file to insert the vectors at the proper   */
/*                 location in the vector table.                   */
/*-------------------------------------------------------------------*/

  asm("           .global _c_int00        ");
  asm("           .global _c_int08        ");

  asm("           .sect \"vectors\"       ");
  asm("RESET      .word _c_int00 ; external RESET-    ");
  asm("INT0       .word _c_int99 ; external INT0-     ");
  asm("INT1       .word _c_int99 ; external INT1-     ");
  asm("INT2       .word _c_int99 ; external INT2-     ");
  asm("INT3       .word _c_int99 ; external INT3-     ");
  asm("XINT0      .word _c_int99 ; Serial port 0 XMT  ");
  asm("RINT0      .word _c_int99 ; Serial port 0 RCV  ");
  asm("XINT1      .word _c_int99 ; Serial port 1 XMT  ");
  asm("RINT1      .word _c_int08 ; Serial port 1 RCV  ");
  asm("TINT0      .word _c_int99 ; Timer 0            ");
  asm("TINT1      .word _c_int99 ; Timer 1            ");
  asm("DINT       .word _c_int99 ; DMA complete       ");

  asm("           .space 20      ; Reserved space     ");

  asm("TRAP0 ");
  asm("           .loop 28       ; TRAPS 0-27 are     ");
  asm("           .word _c_int99 ; undefined traps    ");
  asm("           .endloop                            ");

  asm("           .space 4       ; TRAPS 28-31 reserved");

/*-------------------------------------------------------------------*/
/* NOTE: Put all interrupt handlers AFTER this next statement!      */
/*                                                                   */
/*-------------------------------------------------------------------*/
  asm("           .text                               ");

void c_int99() { }       /* Spurious interrupt handler */
```